Developer manual

1.5.0

Envira Sostenible S.A.

Nanoenvi IAQ

envira IOT

Table of Contents

Chapter 1. Revision control

Version	Changes
V1.3	Integration guide created
V1.4	Documented noise sensor
V1.5	Added methods to reset, factory reset and put in configuration mode. Errata correction.

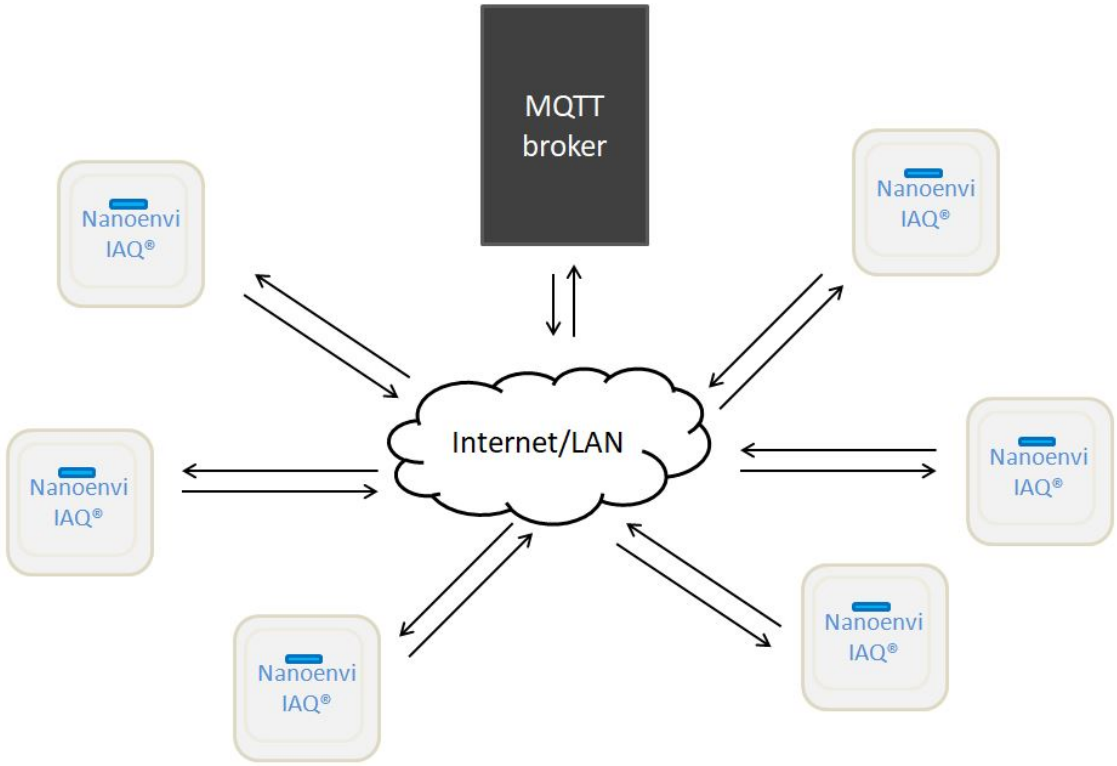
Chapter 2. Scope of the document

This document details aspects of Nanoenvi IAQ™ integration with MQTT broker. Communication interfaces, message format and related aspects are documented.

Chapter 3. Introduction

Nanoenvi IAQ™ has been developed under the #CloudFree philosophy. This device offers open communication and is integrable with data visualization or exploitation platforms without depending on proprietary software or subscriptions to payment services. To do so Nanoenvi IAQ™ uses the MQTT protocol, an open communication protocol oriented towards the transfer of sensor data. Over the last few years became one of the protocols used in IoT systems due to its simplicity, lightness and ease of integration.

The only requirement for Nanoenvi IAQ™ is the use of an MQTT broker; the integrator can therefore consume all data with the freedom to exploit it as desired: either from a BMS or from a data visualization platform.



During its installation, Nanoenvi IAQ™ requires the credentials' configuration of the WiFi network used to send data, the MQTT broker address and access parameters to it. Once configured, Nanoenvi IAQ™ sends the measurements obtained to the MQTT broker and offers an API to facilitate its management, check its status and notify new firmware updates.

For more details on the installation, configuration and operation process, refer to the Nanoenvi IAQ™ user manual

Chapter 4. Interfaces

4.1. Sending sensor measurements via MQTT

Measurement data from the different sensors is sent through the MQTT protocol to the configured broker every T seconds, with T being the established data sending period. The following is the format of the MQTT message received by the broker.

4.1.1. Measurements message format

4.1.1.1. Measurements format

As of firmware version V1.2, sensor measurements are sent to the topic configured in SenML (Sensor Measurement Lists) format as defined by RFC8428 [https://tools.ietf.org/html/rfc8428]. The message format is shown below.

Note that depending on the sensors included in the device, some of the fields are not available. For example, in the case of not having a noise sensor, the corresponding field will not be displayed in the message. Ditto in the case of the absence of a CO sensor.

```
{
    "device_info": {
        "uuid": "<UUID>",
        "fw_ver": "<firmware version>"
    },
    "measures": [
            {
                "bn": "",
                "bver": <Nanoenvi IAQ firmware version>
             },
            {
                "n": "co2",
                "u": "ppm",
                "v": <numerical value of CO2 measurement>
            },
            {
                "n": "voc",
                "u": "ppb",
                "v": <numerical value of VOC measurement>
            },
            {
                "n": "co",
                "u": "ppm",
                "v": <numerical value of CO measurement>
            },
            {
                "n": "noise",
                "u": "%",
                "v": <numerical value of noise level>
            },
            {
                "n": "pm10",
                "u": "ug/m3",
                "v": <numerical value of PM10 measurement>
            },
            {
                "n": "pm2.5",
                "u": "ug/m3",
                "v": <numerical value of PM2.5 measurement>
            },
            {
                "n": "temp",
                "u": "Cel",
                "v": <numerical value of temperature measurement>
```

```
            },
            {
                "n": "hum",
                "u": "%RH",
                "v": <numerical value of humidity measurement>
            },
            {
                "n": "prb",
                "u": "mmHg",
                "v": <numerical value of barometric pressure measurement>
            },
            {
                "n": "pm1",
                "u": "ug/m3",
                "v": <numerical value of PM1 measurement>
            },
            {
                "n": "pm4",
                "u": "ug/m3",
                "v": <numerical value of PM4 measurement>
            },
            {
                "n": "iaqi",
                "u": "count",
                "v": <numerical value of IAQI (Indoor Air Quality Index)>
            },
            {
                "n": "tci",
                "u": "count",
                "v": <numerical value of TCI (Thermal Comfort Index)>
            },
            {
                "n": "eiaqi",
                "u": "count",
                "v": <numerical value of EIAQI (Environmental Indoor Air Quality
                      Index)>
            }
        ]
    }
```

The message contains the UUID and the firmware version of the device. The entries for each measurement contain the abbreviated name of the measured variable, the measurement units as defined in RFC8428 [https://tools.ietf.org/html/rfc8428] and the value of the measurement.

The measurements of particulate matters are presented cumulatively by default. That is, the PM10 concentration value includes the concentration of particles with smaller sizes (PM4, PM2.5 and PM1), the PM4 concentration value includes the concentration of PM2.5 and PM1 and PM2.5 also includes PM1.

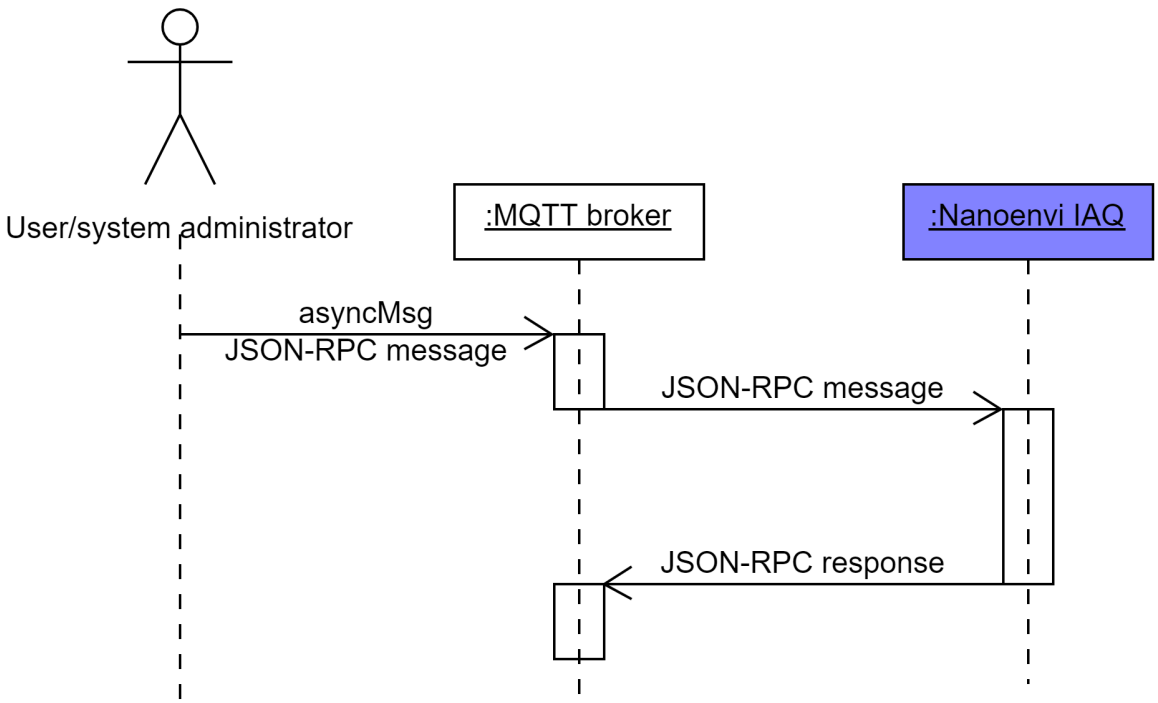
If you want these measurements to be given by interval (not cumulatively), you must send a configuration message to the device (see documentation on JSON-RPC).

4.2. Remote configuration API

4.2.1. Remote configuration operation

When Nanoenvi IAQ™ is in communication mode, it subscribes to the topic `IAQ_RPC/<UUID>` in the configured MQTT broker. Through this topic, the device can receive configuration messages. The responses to these messages are published in the topic `IAQ_RESP/<UUID>`.

The following image describes the message flow chart between Nanoenvi IAQ™ and the MQTT broker, when the user or system administrator launches remote configuration messages:



The configuration message format is based on the standard JSON-RPC 1.0 [https://www.jsonrpc.org/specification_v1], with the following modifications:

- An invalid response/request does not close the communication.
- The JSON-RPC message payload size must not exceed 982 bytes.
- Responses larger than 937 bytes will not be returned.
- Notification messages receive a reply.

The modifications mentioned above are intended to add security and avoid possible memory overflows due to sending messages too long or a number of requests too high.

Within the device, received messages are stored in a queue and processed and responded to as soon as possible in the order they are received.

Note that messages are processed on a first-come, first-served basis.

Additionally, some messages (such as the WiFi configuration change) may cause the Nanoenvi IAQ™ to restart. This can cause the loss of messages received later.

It is important that you also consider the following points:

- If a message is too long or the device's message queue is full, it responds with an error message.
- If many config read methods are invoked in short periods of time, some of the responses may not be received (this could happen if the device's output buffer fills up).
- Some methods involve restarting the device, so please note that all messages received thereafter will be ignored until the device restarts.

4.2.2. Public methods

- **rpc.list**: allows you to list the displayed methods.
- **config.get.aqi_params**: allows you to read the configuration of parameters related to air quality indices such as the measurements considered for the IAQI calculation or which parameter is shown in the LEDs status (TCI, IAQI or EIAQI).
- **config.set.aqi_params**:allows to change the configuration of parameters related to air quality indices such as the measurements considered for the IAQI calculation or which parameter is shown in the LEDs status (TCI, IAQI or EIAQI).
- **config.get.sensor_params**: allows to read the operating parameters of configured sensors such as, for example, the self-calibration of the CO2 sensor.
- **config.set.sensor_params**: allows you to change the operating parameters of configured sensors, such as, for example, the self-calibration of the CO2 sensor.
- **co2.frc:** allows you to force the calibration of the CO2 sensor, sending a known CO2 value.
- **config.get.measures_coefcs**: allows you to read the correction factors applied to measurements.
- **config.set.measures_coefcs**: allows to change the correction factors applied to measurements.
- **config.get.mqtt**: allows you to read the MQTT configuration and the data sending period.
- **config.set.mqtt**: allows you to change the MQTT configuration and the data sending period.
- **config.get.wifi**: returns the SSID of the WiFi to which the device connects.
- **config.set.wifi**: changes the device's WiFi settings.
- **device_info**: displays the MAC address and the firmware version of the device.
- **ping**: you to know if a device is operational and the connection with the MQTT broker is correct.
- **download_ota**: notifies the device of an available update.
- **reset:** reboots the device.
- **factory_reset**: resets the Nanoenvi IAQ™ settings to factory settings.

4.2.3. Public methods description

4.2.3.1. rpc.list

Description:

The `rpc.list` method sends a message with the JSON-RPC methods exposed by Nanoenvi IAQ™.

Format:

```
{
    "id": <message id>,
    "method": "rpc.list"
}
```

Example:

```
{
```

```
    "id": 3,
    "method": "rpc.list"
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": [<list of public JSON-RPC methods available>]
}
```

In case of an incorrect message or lack of space in the device's message queue, an error message will be displayed (see documentation for error messages).

4.2.3.2. download_ota

Description:

The `download_ota` method alerts Nanoenvi IAQ™ to download an update of the OTA firmware.

After receiving the message, the device will verify that the update binary is compatible with the device's CPU. This process is carried out by checking the name of it. If the name of the binary indicates that it's not supported by the CPU, the OTA update is discarded.

Format:

```
{
    "id": <message id>,
    "method": "download_ota",
    "params": {
        "url": <firmware binary URL>,
        "hash": <firmware binary MD5 hash>
    }
}
```

Example:

```
{
    "id": 1,
    "method": "download_ota",
    "params": {
        "url": "http://192.168.137.1:8080/firmware_binary.bin",
        "hash": "e44a5eea993e8a34e938feab4c2f"
    }
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": "OTA notified successfully"
}
```

In case of an incorrect message or lack of space in the device ‘s message queue, an error message will be displayed (see documentation for error messages).

In the event that the name of the update binary indicates that it's not compatible with the CPU, the message displayed will be the following:

```
{
    "id":<message id>,
    "error": {
        "code": -32602,
        "message": "download_ota",
        "data": "OTA update suspected to be incompatible with CPU"
    }
}
```

4.2.3.3. ping

Description:

The `ping` method is sent to the Nanoenvi IAQ™ as a ping to verify that the device is connected and available.

Format:

```
{
    "id": <message id>,
    "method": "ping"
}
```

Example:

```
{
    "id": 1,
    "method": "ping"
}
```

Response:

If the received message is correct and there is space in the device’s message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": "ACK"
}
```

In case of an incorrect message or lack of space in the device's message queue, an error message will be displayed (see documentation for error messages).

4.2.3.4. device_info

Description:

The `device_info` method is sent to request metadata from the device. In this version, the following data are displayed:

- WiFi module MAC address
- Device IP

- The RSSI value is read only at Nanoenvi IAQ™ startup.

The RSSI value is read when Nanoenvi IAQ™ starts up, so if you want to know the real/current value of it, it is advisable to previously reboot the device.

- Microcontroller ID.
- Firmware version.

Format:

```
{
    "id": <message id>,
    "method": "device_info"
}
```

Example:

```
{
    "id": 1,
    "method": "device_info"
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{"id":<message id>,"result":
{
   "mac_adress":"<MAC address>",
   "ip_address":"<firmware version>",
   "rssi":<WiFi signal strength in dBm>,
   "fw_ver":"<firmware version>",
   "cpu":"<microcontroller device signature as string representing device
          signature in hexadecimal format>"
}}
```

In case of an incorrect message or lack of space in the device ‘s message queue, an error message will be displayed (see documentation for error messages).

4.2.3.5. config.set.wifi

Description:

The `config.set.wifi` method allows you to send the WiFi configuration parameters.

After the method is sent, the device will restart and the messages received afterward may be lost.

After restarting, the device will connect to the configured WiFi network, so if this is not available or if an incorrect network was configured, communication with it will be lost.

Format:

```
{
    "id": <message id>,
    "method": "config.set.wifi",
    "params": {
        "ssid": "<SSID of the new WiFi network (string, max 32 characters)>",
        "pass": "<string, WiFi network password (string, max 63 characters)>"
    }
}
```

The message must include all the fields (SSID and password) for it to be taken into consideration.

Example:

```
{
    "id": 1,
    "method": "config.set.wifi",
    "params": {
        "ssid": "SSID_EXAMPLE",
        "pass": "PASSWORD"
    }
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": "wifi config updated"
}
```

In case of an incorrect message or lack of space in the device 's message queue, an error message will be displayed (see documentation for error messages).

4.2.3.6. config.get.wifi

Description:

The `config.get.wifi` method allows you to query the WiFi settings of Nanoenvi IAQ™. Only the name of the network to which Nanoenvi IAQ™ is connected will be displayed.

Format:

```
{
    "id": <message id>,
    "method": "config.get.wifi"
}
```

Example:

```
{
    "id": 1,
    "method": "config.get.wifi"
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": {
        "ssid": "<SSID of the new WiFi network>"
    }
}
```

In case of an incorrect message or lack of space in the device 's message queue, an error message will be displayed (see documentation for error messages).

4.2.3.7. config.set.mqtt

Description:

The `config.set.mqtt` method allows you to change the device's MQTT connection settings remotely.

After the method is sent, the device will restart and the messages received afterward may be lost.

After rebooting, the device will connect to the MQTT broker with the new configured parameters, so if it is not available or if it was configured incorrectly, communication with the device will be lost.

Format:

```
{
    "id": <message id>,
    "method": "config.set.mqtt",
    "params": {
        "host": "<MQTT broker URL (string, 31 characters max)>",
        "port": <port used to connect to MQTT broker (integer, max. 65535)>,
        "ssl": <indicates if connection is secure or not (boolean: true or false)>,
        "comm_period": <communication period in seconds(integer,4 digits max.)>,
        "user": "<MQTT user (string, max 16 characters)>",
        "pass": "<MQTT password (string, max 16 characters)>",
        "topic": "<MQTT topic where data measurements are received (string, max. 64
 characters)>"
    }
}
```

The message must include all the fields for it to be taken into consideration.

Example:

```
{
    "id": 1,
    "method": "config.set.mqtt",
    "params": {
        "host": "mqtt-broker-example-name",
        "port": 1883,
        "ssl": false,
        "comm_period": 1200,
        "user": "my_mqtt_user",
        "pass": "my_mqtt_password",
        "topic": "sensor_data_measures"
    }
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": "mqtt config updated"
}
```

In case of an incorrect message or lack of space in the device ‘s message queue, an error message will be displayed (see documentation for error messages).

4.2.3.8. config.get.mqtt

Description:

The `config.get.mqtt` method allows you to read the MQTT configuration of the device.

Format:

```
{
    "id": <message id>,
    "method": "config.get.mqtt"
}
```

Example:

```
{
    "id": 1,
    "method": "config.get.mqtt"
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id": <message id>,
    "method": "config.set.mqtt",
    "result": {
        "broker": "<MQTT broker URL (string, 31 characters max)>",
        "port": <port used to connect to MQTT broker (integer,4 digits max.)>,
        "ssl": <indicates if connection is secure or not (boolean: true or false)>,
        "comm_period": <communication period in seconds(integer,4 digits max.)>,
        "user": "<MQTT user (string, max 16 characters)>",
        "pass": "<MQTT password (string, max 16 characters)>",
        "topic": "<MQTT topic where data measurements are received (string, max. 64
 characters)>"
    }
}
```

In case of an incorrect message or lack of space in the device ‘s message queue, an error message will be displayed (see documentation for error messages).

4.2.3.9. config.set.measures_coefcs

Description:

The `config.set.measures_coefcs` method allows you to remotely change the linear adjustment values of the device’s sensor measurements.

After acquiring data from the sensors, Nanoenvi IAQ™ applies a linear adjustment that allows compensation to be made on the measured value:

$$corrected\_measure = a * sensor\_measure + b \quad (4.1)$$

Note that if a=1 and b=0, corrected_measure=sensor_measure (compensation disabled).

Although it's allowed to apply this compensation for all measurements, it does not make sense to do so in the following cases (for which the coefficient a must be 1 and the coefficient b must be 0):

- CO2: if automatic calibration is active. In this case, parameters a and b lose their validity when the sensor self-calibrates, leading to erroneous measurements.
- VOC: The VOC sensor implements a self-calibration, so the linear adjustment parameters lose their validity each time a self-calibration is performed.

Format:

```
{
    "id": <message id>,
    "method": "config.set.measures_coefcs",
    "params": {
        "co2": {
            "a": <co2 measurement compensation slope, (float)>,
            "b": <co2 measurement compensation offset, (float)>
        },
        "voc": {
            "a": <voc measurement compensation slope, (float)>,
            "b": <voc measurement compensation offset, (float)>
        },
        "co": {
            "a": <co measurement compensation slope, (float)>,
            "b": <co measurement compensation offset, (float)>
        },
        "pm1": {
            "a": <pm1 measurement compensation slope, (float)>,
            "b": <pm1 measurement compensation offset, (float)>
        },
        "pm2_5": {
            "a": <pm2.5 measurement compensation slope, (float)>,
            "b": <pm2.5 measurement compensation offset, (float)>
        },
        "pm4": {
            "a": <pm4 measurement compensation slope, (float)>,
            "b": <pm4 measurement compensation offset, (float)>
        },
        "pm10": {
            "a": <pm10 measurement compensation slope, (float)>,
            "b": <pm10 measurement compensation offset, (float)>
        },
        "t": {
            "a": <temperature measurement compensation slope, (float)>,
            "b": <temperature measurement compensation offset, (float)>
        },
        "rh": {
            "a": <relative humidity measurement compensation slope, (float)>,
            "b": <relative humidity measurement compensation offset, (float)>
        },
        "noise": {
            "a": <noise level measurement compensation slope, (float)>,
            "b": <noise level measurement compensation offset, (float)>
```

```
        }
    }
}
```

For the message to be considered correct, it's not necessary to include the adjustment factors of all the measured variables. For each configured variable it's required to define the coefficients a and b. Otherwise, the parameter will not be taken into consideration.

Example:

Example of setting the adjustment factors of all the measured parameters:

```
{
    "id": 3,
    "method": "config.set.measures_coefcs",
    "params": {
        "co2": {
            "a": -111.11111,
            "b": -111.11111
        },
        "voc": {
            "a": -111.11111,
            "b": -111.11111
        },
        "co": {
            "a": -111.11111,
            "b": -111.11111
        },
        "pm1": {
            "a": -111.11111,
            "b": -111.11111
        },
        "pm2_5": {
            "a": -111.11111,
            "b": -111.11111
        },
        "pm4": {
            "a": -111.11111,
            "b": -111.11111
        },
        "p": {
            "a": -111.11111,
            "b": -111.11111
        },
        "pm10": {
            "a": -111.11111,
            "b": -111.11111
        },
        "t": {
            "a": -111.11111,
            "b": -111.11111
        },
        "rh": {
            "a": -111.11111,
            "b": -111.11111
        },
        "noise": {
            "a": -111.11111,
            "b": -111.11111
        }
    }
}
```

Example of setting the adjustment factors of temperature measurements:

```
{
```

```
    "id": 3,
    "method": "config.set.measures_coefcs",
    "params": {
        "t": {
            "a": 1,
            "b": -2.0
        }
    }
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": "updated : [<name of measurement parameters which factors have been updated>]"
}
```

In case of an incorrect message or lack of space in the device 's message queue, an error message will be displayed (see documentation for error messages).

4.2.3.10. config.get.measures_coefcs

Description:

The `config.get.measures_coefcs` method allows you to obtain the linear compensation parameters of the measurements acquired by the different sensors.

Format:

```
{
    "id": <message id>,
    "method": "config.get.measures_coefcs"
}
```

Example:

```
{
    "id": 1,
    "method": "config.get.measures_coefcs"
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id": <message id>,
    "method": "config.get.measures_coefcs",
    "result": {
        "co2": {
            "a": <co2 measurement compensation slope, (float)>,
            "b": <co2 measurement compensation offset, (float)>
        },
        "voc": {
            "a": <voc measurement compensation slope, (float)>,
            "b": <voc measurement compensation offset, (float)>
        },
        "co": {
```

```
            "a": <co measurement compensation slope, (float)>,
            "b": <co measurement compensation offset, (float)>
        },
        "pm1": {
            "a": <pm1 measurement compensation slope, (float)>,
            "b": <pm1 measurement compensation offset, (float)>
        },
        "pm2_5": {
            "a": <pm2.5 measurement compensation slope, (float)>,
            "b": <pm2.5 measurement compensation offset, (float)>
        },
        "pm4": {
            "a": <pm4 measurement compensation slope, (float)>,
            "b": <pm4 measurement compensation offset, (float)>
        },
        "pm10": {
            "a": <pm10 measurement compensation slope, (float)>,
            "b": <pm10 measurement compensation offset, (float)>
        },
        "t": {
            "a": <temperature measurement compensation slope, (float)>,
            "b": <temperature measurement compensation offset, (float)>
        },
        "p": {
            "a": <pressure measurement compensation slope, (float)>,
            "b": <pressure measurement compensation offset, (float)>
        },
        "rh": {
            "a": <relative humidity measurement compensation slope, (float)>,
            "b": <relative humidity measurement compensation offset, (float)>
        },
        "noise": {
            "a": <noise level measurement compensation slope, (float)>,
            "b": <noise level measurement compensation offset, (float)>
        }
    }
}
```

In case of an incorrect message or lack of space in the device's message queue, an error message will be displayed (see documentation for error messages).

4.2.3.11. config.set.sensor_params

Description:

The `config.set.sensor_params` method allows you to remotely change operating parameters of different sensors. Specifically, it allows you to change the operation options listed below:

- CO2:
  - enable/disable auto calibration.
  - enable/disable measurement compensation based on atmospheric pressure.
- PM:
  - flag to activate/deactivate cumulative PM measurements. By default, PM measures are returned cumulatively. For example, the PM10 particle concentration includes all particles that are less than 10 microns in size, PM4 all particles that are less than 4 microns, etc.

    If this option is disabled, the measurements of concentrations of suspended particles will be given by interval: For example, the concentration of PM10 particles, includes only the particles whose size is between 10 and 4 microns, PM4 includes only the particles whose size is between 4 and 2.5 microns, etc.

Format:

```
{
    "id": <message id>,
    "method": "config.set.sensor_params",
    "params": {
        "co2": {
            "autocalibration_active": <autocalibration active or not (boolean: true or
 false)>,
            "pres_compensation": <pressure compensation active or not (boolean: true or
 false)>
        },
        "pm": {
            "cumulative_reads": <boolean value to indicate whether pm measurements are
 cumulative or not (boolean: true or false)>
        }
    }
}
```

For the message to be considered correct, it's not necessary to include entries corresponding to all the sensors.

Example:

```
{
    "id": 3,
    "method": "config.set.sensor_params",
    "params": {
        "co2": {
            "autocalibration_active": true,
            "pres_compensation": true
        },
        "pm": {
            "cumulative_reads": false
        }
    }
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": {
        "updated": [<list of sensors which compensation factors have been updated
 successfully>]
    }
}
```

In case of an incorrect message or lack of space in the device 's message queue, an error message will be displayed (see documentation for error messages).

4.2.3.12. config.get.sensor_params

Description:

The `config.set.sensor_params` method allows to remotely obtain operating parameters of different sensors.

Format:

```
{
```

```
    "id": <message id>,
    "method": "config.get.sensor_params"
}
```

Example:

```
{
    "id": 8,
    "method": "config.get.sensor_params"
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": {
        "co2": {
            "autocalibration_active": <autocalibration active or not (boolean: true or
 false)>,
            "pres_compensation": <pressure compensation active or not (boolean: true or
 false)>
        },
        "pm": {
            "cumulative_reads":<boolean value to indicate whether pm measurements are
 cumulative or not (boolean: true or false)>
        }
    }
}
```

In case of an incorrect message or lack of space in the device ‘s message queue, an error message will be displayed (see documentation for error messages).

4.2.3.13. co2.frc

Description:

The `co2.frc` method will force the CO2 sensor calibration by sending a known reference value. The value must be expressed in ppm and be between 400 and 2000 ppm.

Format:

```
{
    "id": <message id>,
    "method": "co2.frc",
    "params": {
        "c_ref_ppm":<CO2 reference concentration in ppm>
    }
}
```

Example:

```
{
    "id": <message id>,
    "method": "co2.frc",
    "params": {
        "c_ref_ppm": 550
    }
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": {
        "updated": [<list of sensors which compensation factors have been updated
 successfully>]
    }
}
```

In case of an incorrect message or lack of space in the device ‘s message queue, an error message will be displayed (see documentation for error messages).

4.2.3.14. config.set.aqi_params

Description:

The `config.set.aqi_params` method allows you to configure the parameters related to the calculation of the AQI (Air Quality Index):

- Which AQI parameter is shown in the status lights:
  - Indoor Air Quality Index (IAQI)
  - Thermal Comfort Index (TCI)
  - a combination of the previous two (EIAQI)
- What measures are used to calculate the IAQI. For the calculation of air quality, you can select the concentration measurements of:
  - CO
  - CO2
  - VOC
  - PM10

  This way, you can select which of these variables are included in the IAQI calculation and which are not.

An application example of using this message is setting the Nanoenvi IAQ™ to display the CO2 level through the status lights.

Format:

```
{
    "id": <message id>,
    "method": "config.set.aqi_params",
    "params": {
        "aqi_displayed": "<type of AQI indedex shown by status LEDs: 'iaqi', 'tci', 'eiaqi'>",
        "iaqi_used_measures": [<array of names of measurements used to calculate IAQI, values:
 "co2","co","voc","pm10">]
    }
}
```

Example:

```
{
    "id": 8,
    "method": "config.set.aqi_params",
    "params": {
        "aqi_displayed": "iaqi",
        "iaqi_used_measures": [
            "co2"
        ]
    }
}
```

In this example, the device is configured so that the status lights show the measured CO2 level: the iaqi_used_measures parameter is configured so that the indoor air quality index (IAQI) is calculated using only the CO2 measurement and the aqi_displayed parameter set to "iaqi" so that the indoor air quality index is displayed on the status lights. This shows the CO2 level in the status lights.

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": "aqi parameters updated"
}
```

In case of an incorrect message or lack of space in the device's message queue, an error message will be displayed (see documentation for error messages).

4.2.3.15. config.get.aqi_params

Description:

The `config.set.aqi_params` method allows you read the configuration of the parameters related to the calculation of the AQI (Air Quality Index):

- Which AQI parameter is shown in the status lights:
  - Indoor Air Quality Index (IAQI)
  - Thermal Comfort Index (TCI)
  - a combination of the previous two which results in the Environmental Air Quality Index (EIAQI)
- What measures are used to calculate the IAQI. For the calculation of air quality, you can select the concentration measurements of:
  - CO
  - CO2
  - VOC
  - PM10

Format:

```
{
    "id": <message id>,
```

```
    "method": "config.get.aqi_params"
}
```

Example:

```
{
    "id": 6,
    "method": "config.get.aqi_params"
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<message id>,
    "result": {
        "aqi_displayed": "<type of AQI index shown by status LEDs: 'iaqi','tci','eiaqi'>",
        "iaqi_used_measures": [<array of names of measurements used to calculate IAQI, values:
 'co2','co','voc','pm10'>]
    }
}
```

In case of an incorrect message or lack of space in the device 's message queue, an error message will be displayed (see documentation for error messages).

4.2.3.16. reset

Description:

The `reset` method restarts the Nanoenvi IAQ™ device.

After using this method, the device will restart and the messages received afterward may be lost.

Format:

```
{
    "id": <id de mensaje>,
    "method": "reset"
}
```

Example:

```
{
    "id": 3,
    "method": "reset"
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
```

```
    "id":<id de mensaje>,
    "result": "device will be reset"
}
```

In case of an incorrect message or lack of space in the device 's message queue, an error message will be displayed (see documentation for error messages).

4.2.3.17. factory_reset

Description:

The `factory_reset` method restores the Nanoenvi IAQ™ device settings to factory defaults.

After using this method, the device will restart and the messages received afterward may be lost.

The device will go back to factory settings. This implies that you will lose the connection with the MQTT broker and WiFi network, you will enter configuration mode (it will need to be configured from the configuration Web page from a point near the device) and you will lose the calibration and sensor use parameters.

Format:

```
{
    "id": <id de mensaje>,
    "method": "factory_reset"
}
```

Example:

```
{
    "id": 3,
    "method": "factory_reset"
}
```

Response:

If the received message is correct and there is space in the device's message queue, the response will have the following format:

```
{
    "id":<id de mensaje>,
    "result": "device will be reset to factory settings"
}
```

In case of an incorrect message or lack of space in the device 's message queue, an error message will be displayed (see documentation for error messages).

Nanoenvi